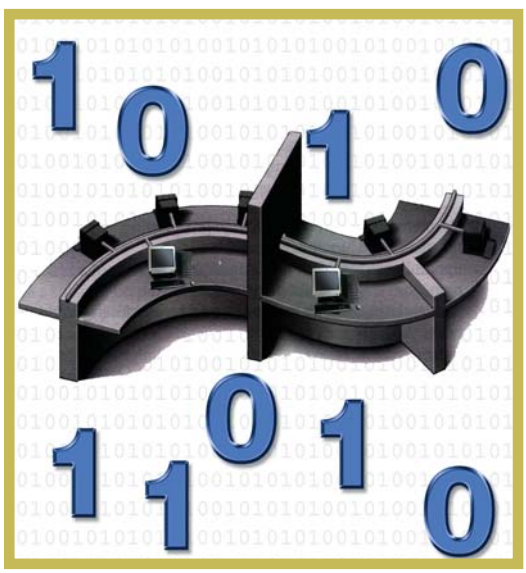
Create Your Own Packet Sniffer

Considered as LAN's best friends, packet sniffers are the protocol analysers meant for data communication networks. This article presents an insight into writing your own packet sniffer.



Packet sniffers are programs which monitor the network traffic that passes through your computer. A packet sniffer that runs on your PC and is connected to the Internet via a modem can tell you your current IP address as well as the IP addresses of the Web servers whose sites you are visiting. You can also watch all the unencrypted data that travels from your computer onto the Internet. This includes passwords and other sensitive data that are not secured by encryption. Put a packet sniffer on a router on the Internet, and you can watch all the network traffic that passes through that router. This includes absolutely anyone whose data happens to pass through that router.

A packet sniffer can grab network packets flowing through your network. In this article, we will show you how to implement your own packet sniffer using the Linux Packet Interface and the freely available Packet Capture Library (*libpcap)*.

Packet sniffers exist in both commercial and in open source forms, and have been widely used to monitor network traffic—in fault analysis, performance analysis, intrusion detection and protocol analysis. They are also used by hackers to steal clear-text passwords and other information from the network.

The sniffing capability of a packet sniffer depends on its position in the network. In general, the packet sniffer should be placed at a point in the network where it can see all the network traffic. In a LAN environment, running the packet sniffer on any Linux box should be sufficient. On the other hand, using a packet sniffer in a switched network may involve some additional configurations on the switch device. Since in a switched network not all switch ports see all the traffic, the switch has to be explicitly configured to mirror all the traffic to the port to which you connect your Linux box running the packet sniffer. You can read your switch device manual to know more about mirroring.

A packet sniffer can either use the Linux Packet Interface or *libpcap* to grab packets from the network. The first section of this article will show how you could develop a packet sniffer with the Linux Packet Interface, and the next section will show you the *libpcap* version.

Using Linux Packet Interface

You can grab packets from the network at the data link layer using the PF_PACKET socket domain type. PF_PACKET can be used to send and receive raw packets from the network. This allows you to implement a protocol stack in the user space.

```
65  int main(int argc, char *argv[])
66  {
67      int pkt_socket = 0, flags = MSG_TRUNC, pkt_len = 0;
68      struct sockaddr_ll from, bindto;
69      socklen_t fromlen;
70      char packet[BUF_SIZE];
71      struct ifreq ifr;
72      struct packet_mreq mr;
73
74      if( argv[1] == NULL ) {
75          printf("\nUsage: %s <device>\n",argv[0]);
76          return 1;
77      }
78
79      pkt_socket = socket(PF_PACKET,SOCK_RAW,htons(ETH_P_IP));
80      if( pkt_socket == -1 ) {
81          perror("socket");
82          return 1;
83      }
84      printf("Socket created.\n");
85
```

Figure 1: Sample program using the Linux Packet Interface to grab packets from the network

Figure 1 shows the snip from a sample program (see listing 1 for complete source) that uses the Linux Packet Interface to grab packets from the network.

You can see (in line 79) that a raw socket of domain type PF_PACKET and socket type SOCK_RAW is created to grab raw packets, including the link level header from the network at the data link layer. Setting the protocol to ETH_P_IP causes the socket to receive all IP packets destined to the host. It should be noted that, by default, only packets that are destined to your host will be seen by the socket. If you want your program to see all packets that traverse your network for some reason, then you need to explicitly configure your interface to which you bind the socket to operate in promiscuous mode.

```
86      memset(&ifr,0,sizeof(ifr));
87      strncpy(ifr.ifr_name,argv[1],sizeof(ifr.ifr_name));
88      if( ioctl(pkt_socket,SIOCGIFINDEX,&ifr) == -1 ) {
89          perror("ioctl");
90          return 1;
91      }
92      printf("Interface index: %d\n",ifr.ifr_ifindex);
93
94      memset(&bindto,0,sizeof(bindto));
95      bindto.sll_family = AF_PACKET;
96      bindto.sll_ifindex = ifr.ifr_ifindex;
97      bindto.sll_protocol = htons(ETH_P_IP);
98      if( bind(pkt_socket,(struct sockaddr *) &bindto,sizeof(bindto)) == -1 ) {
99          perror("bind");
100         return 1;
101     }
102     printf("Socket binded to interface %s\n",argv[1]);
103
104     memset(&mr,0,sizeof(mr));
105     mr.mr_ifindex = ifr.ifr_ifindex;
106     mr.mr_type = PACKET_MR_PROMISC;
107     if( setsockopt(pkt_socket,SOL_PACKET,
108         PACKET_ADD_MEMBERSHIP,&mr,sizeof(mr)) == -1 ) {
109         perror("setsockopt");
110         return 1;
111     }
112     printf("Interface %s set to promiscuous mode.",argv[1]);
113
```

Figure 2: Configuration of interface to operate in promiscuous mode

In Figure 2 you can see that the program first tries to get the index number for the interface to which the socket needs to bind. It then binds the socket to the interface and sets the sniffing mode to promiscuous. This causes your network interface card to accept all packets—even those that are not destined to your host.

In Figure 3 you can see that the *recvfrom* call is used to read packets from the socket. When the *from* argument is not null, then it is filled with the source address of the packet when the *recvfrom* returns. Figure 4 shows the section of the code that displays the packet to the user in readable format.

```
114     while(1) {
115         fromlen = sizeof(from);
116         pkt_len = recvfrom(pkt_socket,packet,BUF_SIZE,
117                            flags,(struct sockaddr *)&from,&fromlen);
118         if( pkt_len == -1 ) {
119             perror("recvfrom");
120             break;
121         }
122
123         display_packet(pkt_len,packet);
124     }
125
126     close(pkt_socket);
127     return 0;
128 }
129
```

Figure 3: Section of code that receives the packet sniffer from the socket

```
17  #define PRINT_IP(addr) \
18      ((unsigned char *) &addr)[0], ((unsigned char *) &addr)[1], \
19      ((unsigned char *) &addr)[2], ((unsigned char *) &addr)[3]
20
21  #define PRINT_MAC(addr) \
22      for( i=0; i<ETH_ALEN; i++ ) { \
23          printf("%02X",addr[i]); \
24          if( (i + 1) < ETH_ALEN ) { \
25              printf(":"); \
26          } \
27      } \
28
29
30  #define BUF_SIZE 1500
31
32  void display_packet(int pkt_len, char *packet)
33  {
34      struct ethhdr *ethernet = NULL;
35      struct iphdr *ip = NULL;
36      static long pkt = 0;
37      int i = 0;
38      time_t tm;
39      char tbuf[256];
40
41      ethernet = (struct ethhdr *) packet;
42      if( ntohs(ethernet->h_proto) != ETH_P_IP ) {
43          return;
44      }
45
46      ip = (struct iphdr *) (packet + sizeof(struct ethhdr));
47
48      tm = time(NULL);
49
50      sprintf(tbuf,"%s",ctime(&tm));
51      if( strchr(tbuf,'\n') != NULL ) {
52          *(strchr(tbuf,'\n')) = '\0';
53      }
54      printf("%ld %s ",(pkt + 1),tbuf);
55      printf("(%d) ",pkt_len);
56      printf("%d.%d.%d.%d (",PRINT_IP(ip->saddr));
57      PRINT_MAC(ethernet->h_source);
58      printf(")");
59      printf(" -> %d.%d.%d.%d (",PRINT_IP(ip->daddr));
60      PRINT_MAC(ethernet->h_dest);
61      printf(") Internet Protocol\n");
62      pkt++;
63  }
64
```

Figure 4: Section of the code displaying the packet in readable format

```
# ./sniff

Usage: ./sniff <device>
# ./sniff eth0
Socket created.
Interface index: 2
Socket binded to interface eth0
Interface eth0 set to promiscuous mode.
1 Mon Jul 19 07:04:28 2004 (74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.171
.129 (00:50:56:72:75:E1) Internet Protocol
2 Mon Jul 19 07:04:29 2004 (74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.171
.129 (00:50:56:72:75:E1) Internet Protocol
3 Mon Jul 19 07:04:30 2004 (74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.171
.129 (00:50:56:72:75:E1) Internet Protocol
4 Mon Jul 19 07:04:31 2004 (74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.171
.129 (00:50:56:72:75:E1) Internet Protocol

# _
```

Figure 5: Sample code implemented with Linux Packet Interface

Figure 5 shows the sample run of the packet sniffer implemented with Linux Packet Interface.

Using Packet Capture Library

This section shows how you could implement your own packet sniffer using the freely available Packet Capture Library *(libpcap)*. *Libpcap* provides a system-independent interface to capture packets from user space. It can be used to obtain

information about the interfaces present in your system, sniff and send packets from and to your network, save and load the packet capture files, create filters and apply them to the captured packets. You can download *libpcap* from the website http://www.tcpdump.org/.

The first step in programming *libpcap* is to ask the library to find the right device to sniff. Figure 6 shows the section of code that uses the *pcap_lookupdev* to find the right device (see listing 2 for complete source).

```
74
75  void sig_handler(int signum)
76  {
77      switch(signum) {
78          case SIGINT:
79              exit(0);
80      }
81  }
82
83  void exit_handler()
84  {
85      if( PCAP != NULL ) {
86          pcap_close(PCAP);
87      }
88  }
89
90  int main(int argc, char *argv[])
91  {
92      char *dev = NULL;
93
94      atexit(exit_handler);
95      signal(SIGINT,sig_handler);
96
97      if( (dev=pcap_lookupdev(errbuf)) == NULL ) {
98          perror("pcap_lookupdev");
99          exit(1);
100     }
101
102     sniff(dev);
103     exit(0);
104 }
105
```

Figure 6: Section of code using pcap_lookupdev for finding the right device

Once the right device has been identified, the next step is to open the device for sniffing. The section of code shown in Figure 7 illustrates this.

```
57  void sniff(char *intName)
58  {
59      int ret = 0;
60
61      printf("Sniffing packets from %s\n",intName);
62
63      if( (PCAP=pcap_open_live(intName,65535,1,0,errbuf)) == NULL ) {
64          printf("ERROR: %s\n",errbuf);
65          return;
66      }
67
68      if( (ret=pcap_loop(PCAP,-1,pkt_handler,NULL)) != 0 ) {
69          printf("ret  %d\n",ret);
70          printf("ERROR: pcap_loop failed.\n");
71          return;
72      }
73  }
74
```

Figure 7: Section of code opening the device for sniffing

In Figure 7, you can see that *pcap_open_live* is used to open the device for sniffing. The third parameter to *pcap_open_live* instructs the library to configure the device in promiscuous mode. The *pcap_loop* is used to grab the packets from the device. It uses the callback handler *pkt_handler* to notify the arrival of the packets. Please refer to the man page of *pcap* to know more about the other APIs. Figure 8 shows the implementation of the *pkt_handler* callback routine. All that it does is to display the packet in user readable format.

```
11  #define PRINT_IP(addr) \
12      ((unsigned char *) &addr)[0], ((unsigned char *) &addr)[1], \
13      ((unsigned char *) &addr)[2], ((unsigned char *) &addr)[3]
14
15  #define PRINT_MAC(addr) \
16      for( i=0; i<ETH_ALEN; i++ ) { \
17          printf("%02X",addr[i]); \
18          if( (i + 1) < ETH_ALEN ) { \
19              printf(":"); \
20          } \
21      } \
22
23  char errbuf[PCAP_ERRBUF_SIZE];
24  pcap_t *PCAP = NULL;
25
26  void pkt_handler(u_char *args, const struct pcap_pkthdr *header,
27      const u_char *packet)
28  {
29      struct ethhdr *ethernet = NULL;
30      struct iphdr *ip = NULL;
31      static long pkt = 0;
32      int i = 0;
33      char buf[256];
34
35      ethernet = (struct ethhdr *) packet;
36      if( ntohs(ethernet->h_proto) != ETH_P_IP ) {
37          return;
38      }
39
40      ip = (struct iphdr *) (packet + sizeof(struct ethhdr));
41
42      sprintf(buf,"%s",ctime((const time_t *)&header->ts.tv_sec));
43      if( strchr(buf,'\n') != NULL ) {
44          *(strchr(buf,'\n')) = '\0';
45      }
46      printf("%ld %s ",(pkt + 1),buf);
47      printf("(%d:%d) ",header->caplen,header->len);
48      printf("%d.%d.%d.%d (",PRINT_IP(ip->saddr));
49      PRINT_MAC(ethernet->h_source);
50      printf(")");
51      printf(" -> %d.%d.%d.%d (",PRINT_IP(ip->daddr));
52      PRINT_MAC(ethernet->h_dest);
53      printf(") Internet Protocol\n");
54      pkt++;
55  }
```

Figure 8: Section of code showing implementation of the pkt_handler callback routine

```
# ./pcap_sniff
Sniffing packets from eth0
1 Mon Jul 19 07:18:25 2004 (74:74) 192.168.171.1 (00:50:56:C0:00:08) -> 192.168.
171.129 (00:50:56:72:75:E1) Internet Protocol
2 Mon Jul 19 07:18:25 2004 (74:74) 192.168.171.129 (00:50:56:72:75:E1) -> 192.16
8.171.1 (00:50:56:C0:00:08) Internet Protocol
3 Mon Jul 19 07:18:26 2004 (74:74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.
171.129 (00:50:56:72:75:E1) Internet Protocol
4 Mon Jul 19 07:18:26 2004 (74:74) 192.168.171.129 (00:50:56:72:75:E1) -> 192.16
8.171.1 (00:50:56:C0:00:08) Internet Protocol
5 Mon Jul 19 07:18:27 2004 (74:74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.
171.129 (00:50:56:72:75:E1) Internet Protocol
6 Mon Jul 19 07:18:27 2004 (74:74) 192.168.171.129 (00:50:56:72:75:E1) -> 192.16
8.171.1 (00:50:56:C0:00:08) Internet Protocol
7 Mon Jul 19 07:18:28 2004 (74:74) 192.168.171.1 (00:50:56:C0:00:00) -> 192.168.
171.129 (00:50:56:72:75:E1) Internet Protocol
8 Mon Jul 19 07:18:28 2004 (74:74) 192.168.171.129 (00:50:56:72:75:E1) -> 192.16
8.171.1 (00:50:56:C0:00:08) Internet Protocol
# _
```

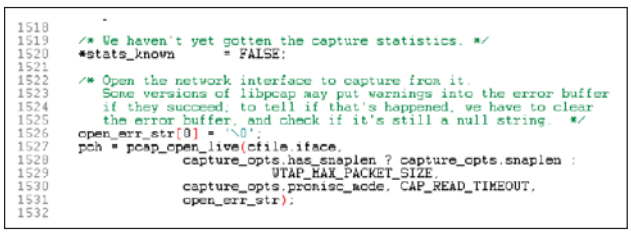
Figure 9: Sample program implemented with libpcap

```
1187  static int
1188  live_open_new(pcap_t *handle, const char *device, int promisc,
1189          int to_ms, char *ebuf)
1190  {
1191  #ifdef HAVE_PF_PACKET_SOCKETS
1192      int          sock_fd = -1, device_id, arptype;
1193      int          err;
1194      int          fatal_err = 0;
1195      struct packet_mreq  mr;
1196
1197      /* One shot loop used for error handling - bail out with break */
1198
1199      do {
1200          /*
1201           * Open a socket with protocol family packet.  If a device is
1202           * given we try to open it in raw mode otherwise we use
1203           * the cooked interface.
1204           */
1205          sock_fd = device ?
1206              socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))
1207                  : socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_ALL));
1208
1209          if (sock_fd == -1) {
1210              snprintf(ebuf, PCAP_ERRBUF_SIZE, "socket: %s",
1211                  pcap_strerror(errno) );
1212              break;
1213          }
```

Figure 10: Section of the libpcap code that is using the Linux Packet Interface

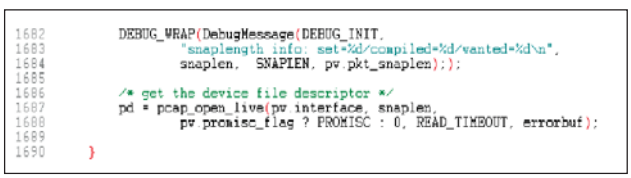
Figure 9 shows the sample run of the packet sniffer implemented with *libpcap*.

Internally, *libpcap* also uses the Linux Packet Interface to grab packets from the network. Figure 10 shows the section of *libpcap* code that is using the Linux Packet Interface.



```
1518
1519  /* We haven't yet gotten the capture statistics. */
1520  *stats_known       = FALSE;
1521
1522  /* Open the network interface to capture from it.
1523     Some versions of libpcap may put warnings into the error buffer
1524     if they succeed; to tell if that's happened, we have to clear
1525     the error buffer, and check if it's still a null string.  */
1526  open_err_str[0] = '\0';
1527  pch = pcap_open_live(cfile.iface,
1528               capture_opts.has_snaplen ? capture_opts.snaplen :
1529                           WTAP_MAX_PACKET_SIZE,
1530               capture_opts.promisc_mode, CAP_READ_TIMEOUT,
1531               open_err_str);
1532
```

Figure 11: Section of code from Ethereal using the libpcap interface



```
1682          DEBUG_WRAP(DebugMessage(DEBUG_INIT,
1683                  "snaplength info: set=%d/compiled=%d/wanted=%d\n",
1684                  snaplen,  SNAPLEN, pv.pkt_snaplen));
1685
1686          /* get the device file descriptor */
1687          pd = pcap_open_live(pv.interface, snaplen,
1688                  pv.promisc_flag ? PROMISC : 0, READ_TIMEOUT, errorbuf);
1689
1690      }
```

Figure 12: Section of code from Snort using the libpcap interface

Figure 11 and 12 show the sections of code, from Ethereal and Snort respectively, which are using the *libpcap* interface.

Listing 1: sniff.c ñPacket Sniffer using Packet Level Interface

```
#include <stdio.h>
#include <errno.h>
#include <netinet/if_ether.h>
#include <netinet/ip.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <features.h>
#include <netpacket/packet.h>
#include <net/ethernet.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <net/if.h>

#define PRINT_IP(addr) \
  ((unsigned char *) &addr)[0], ((unsigned char *) &addr)[1], \
  ((unsigned char *) &addr)[2], ((unsigned char *) &addr)[3]

#define PRINT_MAC(addr) \
  for( i=0; i<ETH_ALEN; i++ ) { \
    printf("%02X",addr[i]); \
    if( (i + 1) < ETH_ALEN ) { \
      printf(":"); \
    } \
  } \


#define BUF_SIZE 1500

void display_packet(int pkt_len, char *packet)
{
  struct ethhdr *ethernet = NULL;
  struct iphdr *ip = NULL;
  static long pkt = 0;
  int i = 0;
  time_t tm;
  char tbuf[256];

  ethernet = (struct ethhdr *) packet;
  if( ntohs(ethernet->h_proto) != ETH_P_IP ) {
    return;
  }
  ip = (struct iphdr *) (packet + sizeof(struct ethhdr));

  tm = time(NULL);

  sprintf(tbuf,"%s",ctime(&tm));
  if( strchr(tbuf,'\n') != NULL ) {
  *(strchr(tbuf,'\n')) = '\0';
  }
  printf("%ld %s ",(pkt + 1),tbuf);
  printf("(%d) ",pkt_len);
  printf("%d.%d.%d.%d (",PRINT_IP(ip->saddr));
  PRINT_MAC(ethernet->h_source);
  printf(")");
  printf(" -> %d.%d.%d.%d (",PRINT_IP(ip->daddr));
  PRINT_MAC(ethernet->h_dest);
  printf(") Internet Protocol\n");
  pkt++;
}

int main(int argc, char *argv[])
{
  int pkt_socket = 0, flags = MSG_TRUNC, pkt_len = 0;
  struct sockaddr_ll from, bindto;
  socklen_t fromlen;
  char packet[BUF_SIZE];
  struct ifreq ifr;
  struct packet_mreq mr;

  if( argv[1] == NULL ) {
    printf("\nUsage: %s <device>\n",argv[0]);
    return 1;
  }

  pkt_socket = socket(PF_PACKET,SOCK_RAW,htons(ETH_P_IP));
  if( pkt_socket == -1 ) {
    perror("socket");
    return 1;
  }
  printf("Socket created.\n");

  memset(&ifr,0,sizeof(ifr));
  strncpy(ifr.ifr_name,argv[1],sizeof(ifr.ifr_name));
  if( ioctl(pkt_socket,SIOCGIFINDEX,&ifr) == -1 ) {
    perror("ioctl");
    return 1;
  }
  printf("Interface index: %d\n",ifr.ifr_ifindex);

  memset(&bindto,0,sizeof(bindto));
  bindto.sll_family = AF_PACKET;
  bindto.sll_ifindex = ifr.ifr_ifindex;
  bindto.sll_protocol = htons(ETH_P_IP);
  if( bind(pkt_socket,(struct sockaddr *) &bindto,sizeof(bindto))
  == -1 ) {
    perror("bind");
    return 1;
  }
  printf("Socket binded to interface %s\n",argv[1]);

  memset(&mr,0,sizeof(mr));
  mr.mr_ifindex = ifr.ifr_ifindex;
  mr.mr_type = PACKET_MR_PROMISC;
  if( setsockopt(pkt_socket,SOL_PACKET,
    PACKET_ADD_MEMBERSHIP,&mr,sizeof(mr)) == -1 ) {
    perror("setsockopt");
    return 1;
  }
  printf("Interface %s set to promiscuous mode.",argv[1]);

  while(1) {
```

```
    fromlen = sizeof(from);
    pkt_len = recvfrom(pkt_socket,packet,BUF_SIZE,
              flags,(struct sockaddr *)&from,&fromlen);
    if( pkt_len == -1 ) {
      perror("recvfrom");
      break;
    }

    display_packet(pkt_len,packet);
  }

  close(pkt_socket);
  return 0;
}
```

Use the following command to compile the previous program...

```
gcc -Wall -g -O -o sniff sniff.c
```

Listing 2: pcap_sniff.c ñPacket Sniffer using Packet Capture Library (libpcap)

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <pcap.h>
#include <signal.h>

#define PRINT_IP(addr) \
  ((unsigned char *) &addr)[0], ((unsigned char *) &addr)[1], \
  ((unsigned char *) &addr)[2], ((unsigned char *) &addr)[3]

#define PRINT_MAC(addr) \
  for( i=0; i<ETH_ALEN; i++ ) { \
    printf("%02X",addr[i]); \
    if( (i + 1) < ETH_ALEN ) { \
    printf(":"); \
    } \
  } \

char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *PCAP = NULL;

void pkt_handler(u_char *args, const struct pcap_pkthdr *header,
  const u_char *packet)
{
  struct ethhdr *ethernet = NULL;
  struct iphdr *ip = NULL;
  static long pkt = 0;
  int i = 0;
  char buf[256];

  ethernet = (struct ethhdr *) packet;
  if( ntohs(ethernet->h_proto) != ETH_P_IP ) {
  return;
  }

  ip = (struct iphdr *) (packet + sizeof(struct ethhdr));

  sprintf(buf,"%s",ctime((const time_t *)&header->ts.tv_sec));
  if( strchr(buf,'\n') != NULL ) {
    *(strchr(buf,'\n')) = '\0';
  }
  printf("%ld %s ",(pkt + 1),buf);
  printf("(%d:%d) ",header->caplen,header->len);
  printf("%d.%d.%d.%d (",PRINT_IP(ip->saddr));
  PRINT_MAC(ethernet->h_source);
  printf(")");
  printf(" -> %d.%d.%d.%d (",PRINT_IP(ip->daddr));
  PRINT_MAC(ethernet->h_dest);
  printf(") Internet Protocol\n");
  pkt++;
}

void sniff(char *intName)
{
  int ret = 0;

  printf("Sniffing packets from %s\n",intName);

  if( (PCAP=pcap_open_live(intName,65535,1,0,errbuf)) == NULL){
    printf("ERROR: %s\n",errbuf);
    return;
  }

  if( (ret=pcap_loop(PCAP,-1,pkt_handler,NULL)) != 0 ) {
    printf("ret: %d\n",ret);
    printf("ERROR: pcap_loop failed.\n");
    return;
  }
}

void sig_handler(int signum)
{
  switch(signum) {
    case SIGINT:
      exit(0);
  }
}

void exit_handler()
{
  if( PCAP != NULL ) {
    pcap_close(PCAP);
  }
}

int main(int argc, char *argv[])
{
  char *dev = NULL;

  atexit(exit_handler);
  signal(SIGINT,sig_handler);

  if( (dev=pcap_lookupdev(errbuf)) == NULL ) {
    perror("pcap_lookupdev");
    exit(1);
  }

  sniff(dev);
  exit(0);
}
```

Use the following command to compile the above program...

```
gcc -Wall -g -O -o pcap_sniff pcap_sniff.c -lpcap
```

That's it! So implementing your own packet sniffer, either using the Linux Packet Interface or the *libpcap*, is no longer a difficult task. Try it out for yourself! LFY

By: Raja R K. The author is a lead engineer working with HCL Technologies (Cisco Systems Offshore Development Centre) in Chennai. He can be contacted at: rajark_hcl@yahoo.co.in